

**FACULTAD DE CIENCIAS E INGENIERÍA**

**CARRERA DE INGENIERÍA DE SOFTWARE**

**TEMA:**Tema de la actividad

**AUTORES:**Nombres del autor o autores

**ASIGNATURA:**Nombre de la asignatura

**DOCENTE:**Nombres del docente

**FECHA DE ENTREGA:**Fecha establecida

**PERIODO:**Abril 2024 a Agosto 2025

**MILAGRO-ECUADOR**

ÍNDICE

[1. Introducción 4](#_Toc214001870)

[2. Objetivos 5](#_Toc214001871)

[3. Desarrollo 5](#_Toc214001872)

[3.1 Configuración de framework 5](#_Toc214001873)

[3.2.1 Entorno virtual 5](#_Toc214001874)

[3.2.2 Creación de un Proyecto Django 6](#_Toc214001875)

[3.2.3 Arquitectura aplicación 8](#_Toc214001876)

[3.2 Configuración base de datos 9](#_Toc214001877)

[3.2.1 Creación de la base de datos y usuario 10](#_Toc214001878)

[3.2.2 Instalación de la librería psycopg2 11](#_Toc214001879)

[3.2.3 Configuración en el archivo settings.py 11](#_Toc214001880)

[3.3 Creación de modelos 12](#_Toc214001881)

[3.3.1 Modelo docentes 12](#_Toc214001882)

[3.3.2 Modelo Materias 13](#_Toc214001883)

[3.4 Migración y ORM 14](#_Toc214001884)

[3.4.1 Agregar aplicaciones en settings 14](#_Toc214001885)

[3.4.2 Generar las migraciones para cada modelo 15](#_Toc214001886)

[3.4.3 Aplicar las migraciones 16](#_Toc214001887)

[3.5 Lógica de negocio 17](#_Toc214001888)

[3.5.1 CRUD docente 17](#_Toc214001889)

[3.5.1.1 Vista Crear 18](#_Toc214001890)

[3.5.1.2 Vista Leer 18](#_Toc214001891)

[3.5.1.3 Vista Actualizar 19](#_Toc214001892)

[3.5.1.4 Vista Eliminar 20](#_Toc214001893)

[3.5.1.5 Formulario para el Modelo 20](#_Toc214001894)

[3.5.2 CRUD Materias 21](#_Toc214001895)

[3.5.2.1 Vista Crear 21](#_Toc214001896)

[3.5.2.2 Vista Leer 22](#_Toc214001897)

[3.5.2.3 Vista Actualizar 22](#_Toc214001898)

[3.5.2.4 Vista Eliminar 23](#_Toc214001899)

[3.5.2.5 Formulario para el Modelo 24](#_Toc214001900)

[3.6 Diseño templates 25](#_Toc214001901)

[3.6.1 Barra de navegación 26](#_Toc214001902)

[3.6.2 Footer 27](#_Toc214001903)

[3.6.4 Docente 29](#_Toc214001904)

[3.6.4.1 Agregar docente 29](#_Toc214001905)

[3.6.4.2 Listar docentes 30](#_Toc214001906)

[3.6.4.3 Editar docente 32](#_Toc214001907)

[3.6.4.4 Eliminar docente 33](#_Toc214001908)

[3.6.5 Materias 34](#_Toc214001909)

[3.6.5.1 Agregar materias 34](#_Toc214001910)

[3.6.5.2 Lista de materias 35](#_Toc214001911)

[3.6.5.3 Editar materia 36](#_Toc214001912)

[3.6.5.4 Eliminar materia 37](#_Toc214001913)

[4. Resultados 38](#_Toc214001914)

[4.1 Navbar 38](#_Toc214001915)

[4.2 Footer 38](#_Toc214001916)

[4.3 Página principal 39](#_Toc214001917)

[4.4 Docentes 39](#_Toc214001918)

[4.5 Materias 42](#_Toc214001919)

[5. Conclusiones 45](#_Toc214001920)

[6. Bibliografía 46](#_Toc214001921)

# Introducción

El desarrollo de aplicaciones web dinámicas constituye una actividad esencial en la ingeniería de software contemporánea, ya que responde a la necesidad de administrar grandes volúmenes de información de manera estructurada, segura y eficiente; en este contexto, los frameworks basados en Python como Django se han consolidado como herramientas robustas para la creación de sistemas escalables, al integrar componentes que automatizan tareas comunes y permiten una arquitectura ordenada fundamentada en el patrón Modelo-Vista-Controlador adaptado al enfoque Modelo-Template-Vista (Sweigart, 2020).

La administración correcta de los datos resulta crítica en plataformas educativas y administrativas, especialmente cuando se busca garantizar integridad, seguridad y trazabilidad; por ello, se seleccionó PostgreSQL como motor de base de datos, reconocido por su estabilidad, capacidad transaccional y soporte para relaciones complejas entre entidades, mientras que la conexión con Django se realizó mediante la librería psycopg2, que permite al ORM mapear los modelos en Python hacia tablas relacionales, facilitando la creación, actualización y consulta de información sin necesidad de manejar directamente instrucciones SQL complejas (Momjian, 2021; Kennedy & Fogel, 2017).

En el sistema desarrollado se implementaron dos aplicaciones principales denominadas Docentes y Materias, cada una con modelos definidos desde el enfoque orientado a objetos, vistas basadas en funciones para gestionar la lógica de negocio y plantillas HTML que aseguran una experiencia visual clara e intuitiva; además, se emplearon formularios tipo ModelForm para automatizar la validación de datos y reforzar la seguridad, evitando inconsistencias en la interacción entre la interfaz y la base de datos, lo que coincide con las buenas prácticas de desarrollo que destacan la reutilización de código y la separación de responsabilidades como pilares para la mantenibilidad del software (Fowler, 2018).

Finalmente, el diseño de la interfaz gráfica se construyó con HTML, CSS y Bootstrap para garantizar una presentación responsiva y adaptable a distintos dispositivos, incorporando elementos como barra de navegación, pie de página, página principal y vistas organizadas mediante tarjetas, lo que refuerza la importancia de ofrecer al usuario una interacción clara y coherente con los principios de usabilidad expuestos en la literatura especializada (Krug, 2014).

# Objetivos

* Configurar el entorno de desarrollo mediante la creación de un proyecto Django y la conexión con la base de datos PostgreSQL utilizando la librería psycopg2.
* Diseñar y desarrollar los modelos de datos para las aplicaciones Docentes y Materias, considerando la relación entre ambas entidades.
* Implementar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) para cada aplicación, garantizando la correcta gestión de la información.
* Diseñar y construir la interfaz de usuario con HTML, CSS (o Bootstrap) y plantillas de Django, asegurando una experiencia clara, funcional y adaptable.
* Aplicar buenas prácticas de desarrollo web utilizando el patrón de diseño MVC y tecnologías como Python, Django y PostgreSQL.

# Desarrollo

## Configuración de framework

### Entorno virtual

Para iniciar el desarrollo, se crea una carpeta denominada P25dw\_app, que servirá como directorio principal del proyecto. Dentro de esta carpeta, se procede a generar un entorno virtual utilizando el comando:

python -m venv venv

Este comando crea una subcarpeta llamada venv, la cual contiene todos los archivos necesarios para gestionar las dependencias del proyecto de manera aislada.

Una vez creado el entorno virtual, se activa mediante el siguiente comando:

venv\Scripts\activate

Al ejecutarlo correctamente, el símbolo del sistema mostrará el prefijo (venv), indicando que el entorno está activo y listo para instalar los paquetes requeridos.

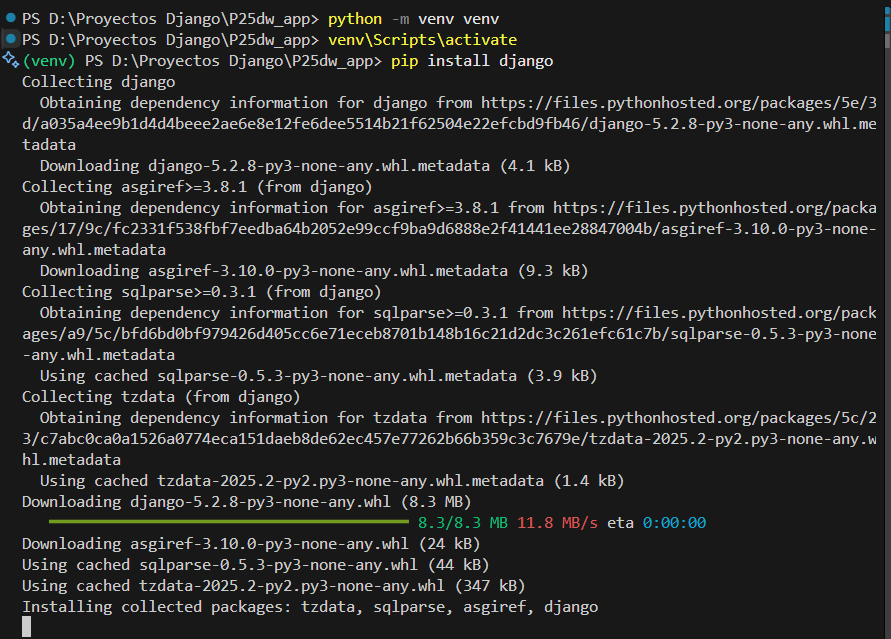
Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 1**: creación del entorno virtual

Con el entorno virtual activado, se procede a instalar el framework Django, el cual será la base para el desarrollo del proyecto. Para ello, se ejecuta el siguiente comando en la terminal::

*pip install django*



**Ilustración 2**: Instalación del framework

### Creación de un Proyecto Django

Una vez instalada la librería Django, se verifica su correcta instalación ejecutando el comando:

django-admin –version

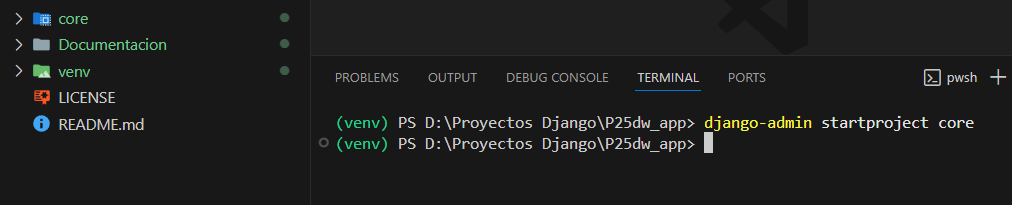
Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 3**: Versión de Django

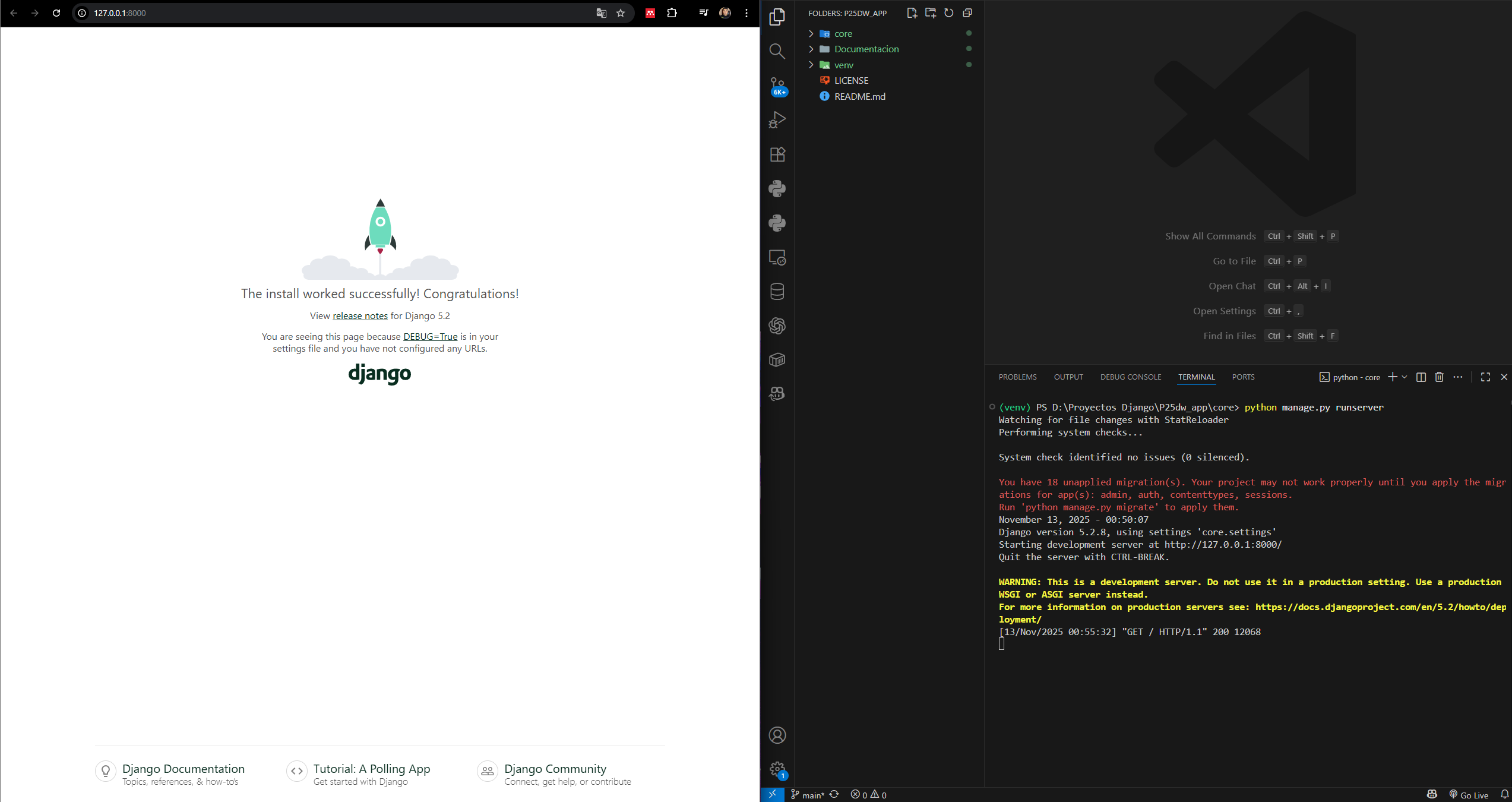
Posteriormente, se procede a crear el proyecto base mediante el comando:

django-admin startproject core



**Ilustración 3**: Creación del proyecto

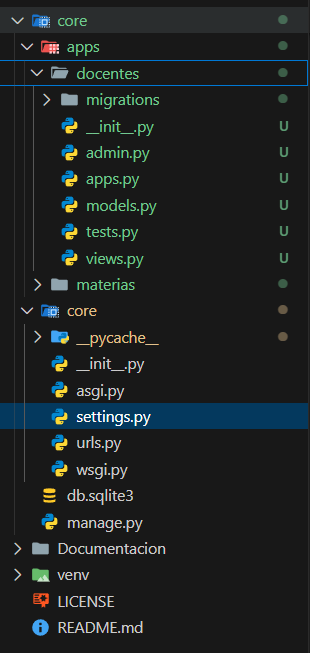
Para confirmar que la instalación de Django se realizó correctamente, se ejecuta el comando django-admin --version, el cual muestra la versión instalada y garantiza que el framework está disponible en el entorno. A continuación, se crea el proyecto base mediante el comando django-admin startproject core, que genera la estructura inicial del proyecto. Después de la creación, se accede al directorio correspondiente con cd P25dw\_app y se inicia el servidor de desarrollo utilizando python manage.py runserver, lo que permite verificar en la consola que el servidor se ha iniciado y acceder al proyecto desde la URL local <http://127.0.0.1:8000>.



**Ilustración 4**: Inicialización del proyecto

### Arquitectura aplicación

La arquitectura implementada en este proyecto se basa en el patrón de diseño MVC (Modelo-Vista-Controlador), adaptado al enfoque propio de Django conocido como MTV (Model-Template-View), lo que permite organizar el código de manera clara, escalable y mantenible mediante la separación de responsabilidades entre los componentes del sistema; la estructura general se organizó creando una carpeta raíz denominada P25dw\_app, dentro de la cual se generó el proyecto principal llamado core que contiene las configuraciones base del framework, y adicionalmente se creó una carpeta apps destinada a almacenar las aplicaciones modulares del sistema, específicamente Docentes y Materia



**Ilustración 5**: Estructura del proyecto

* ***Modelo***

En la capa Modelo se definen las estructuras de datos mediante clases que se declaran en el archivo models.py de cada aplicación. Estas clases representan las entidades del sistema y sus atributos, y posteriormente se transforman en tablas dentro de la base de datos PostgreSQL. Este enfoque permite mantener una relación directa entre el código y la persistencia de datos.

* ***Vista***

Las vistas, en el contexto de Django, se implementan mediante plantillas HTML ubicadas en la carpeta templates/ de cada aplicación. Estas plantillas son responsables de renderizar la información que se presenta al usuario, incluyendo formularios, listados, detalles y mensajes de confirmación, asegurando una interfaz clara y funcional.

* ***Controlador (Views)***

La lógica de negocio se desarrolla en el archivo views.py de cada aplicación. En este componente se implementan las funciones y clases necesarias para gestionar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Las vistas reciben las solicitudes del usuario, interactúan con los modelos y devuelven respuestas renderizadas mediante las plantillas. Además, se definen rutas en urls.py para cada aplicación, las cuales se enlazan con el archivo principal core/urls.py.

* ***Condiguraciones***

El directorio core/ se reserva para la configuración global del proyecto Django. Contiene archivos esenciales como:

*settings.py*: define parámetros generales del proyecto, incluyendo la conexión a la base de datos, aplicaciones instaladas y rutas estáticas.

*urls.py*: gestiona el enrutamiento principal del sistema.

*wsgi.py* y asgi.py: archivos destinados al despliegue en servidores compatibles.

## Configuración base de datos

En este proyecto se ha optado por utilizar el sistema de gestión de bases de datos **PostgreSQL**, el cual ofrece robustez, estabilidad y un sólido soporte para operaciones relacionales. Para la conexión con Django, se utilizó la librería psycopg2, tal como fue enseñado en clases, permitiendo al ORM de Django interactuar con la base de datos de manera transparente y eficiente.

### Creación de la base de datos y usuario

Para garantizar la persistencia de los datos, se procede a la creación de la base de datos en el motor PostgreSQL, junto con el usuario que tendrá los permisos necesarios para su administración.

En primer lugar, se accede a la consola de PostgreSQL y se ejecutan los comandos para crear la base de datos denominada practice25dw y el usuario practiusr, asignándole la contraseña pract2025pwd, posteriormente, se otorgan los privilegios correspondientes para permitir la conexión desde el proyecto Django.

Desde la terminal de PostgreSQL, se ejecutaron los siguientes comandos para crear la base de datos y el usuario correspondiente:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

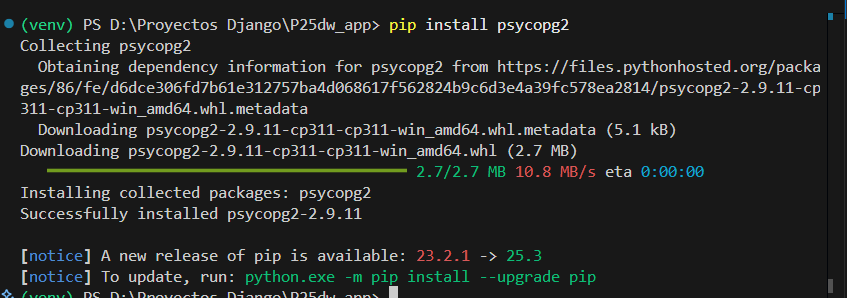
**Ilustración 6**: Creación de la base de datos en Pgadmin

Estos comandos crean la base de datos practice25dw, un usuario denominado practiusr con la contraseña pract2025pwd, y le otorgan todos los privilegios necesarios para que Django pueda operar sobre la base de datos sin restricciones.

### Instalación de la librería psycopg2

Para establecer la conexión entre el proyecto Django y la base de datos PostgreSQL, es necesario instalar la librería psycopg2 dentro del entorno virtual previamente activado. Esta librería actúa como el conector que permite la comunicación entre el framework y el motor de base de datos. La instalación se realiza mediante el siguiente comando ejecutado en la terminal:

pip install psycopg2

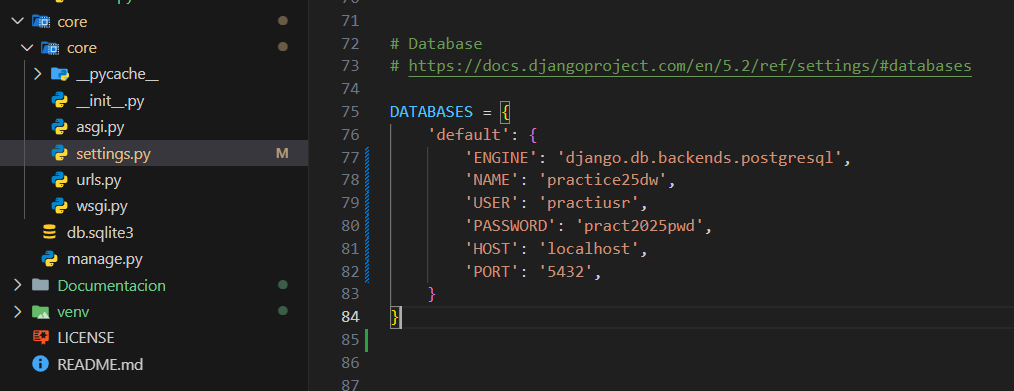


**Ilustración 7**: instalación de la librería ORM psycopg2

### Configuración en el archivo settings.py

Una vez creada la base de datos y el usuario en PostgreSQL, se procede a establecer la conexión en el archivo settings.py del proyecto, ubicado en el directorio core/. En la sección DATABASES, se reemplaza la configuración predeterminada de SQLite por los parámetros correspondientes a PostgreSQL, asegurando que el ORM de Django interactúe directamente con la base de datos definida.

La configuración aplicada es la siguiente:



**Ilustración 8**: Credenciales de acceso en django

Esta estructura permite que todas las operaciones realizadas por el ORM de Django se ejecuten sobre la base de datos PostgreSQL, garantizando la correcta integración entre el framework y el motor de datos.

## Creación de modelos

Cada aplicación del proyecto (docentes y materias) incluye un archivo models.py donde se definen las estructuras de datos que se convertirán en tablas dentro de la base de datos PostgreSQL. Estas clases representan las entidades del sistema y sus atributos, permitiendo que el ORM de Django gestione la persistencia de manera eficiente.

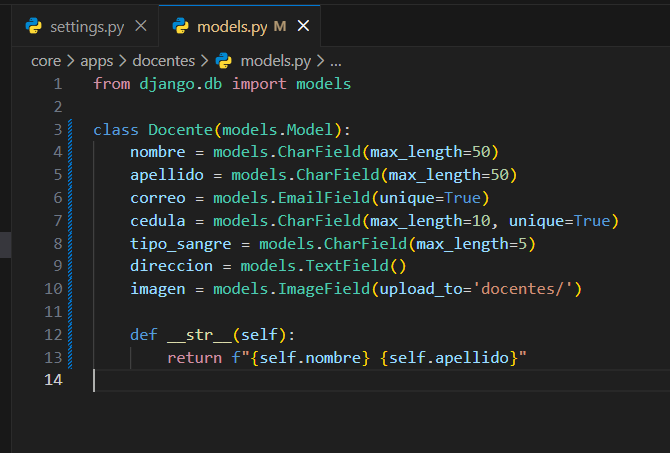
De acuerdo con el enunciado, “Un docente puede impartir varios cursos, y varios cursos pueden ser impartidos por un solo docente”, se implementa una relación muchos a muchos entre las entidades Docente y Curso, utilizando el modelo intermedio que Django proporciona para este tipo de relaciones.

### Modelo docentes

El modelo Docente representa la información del personal académico y contiene los siguientes atributos:

* nombre (CharField)
* apellido (CharField)
* correo (EmailField)
* cédula (CharField)
* tipo\_sangre (CharField)
* dirección (TextField)
* imagen (ImageField)

Este modelo representado en la aplicación se muestra a continuación :



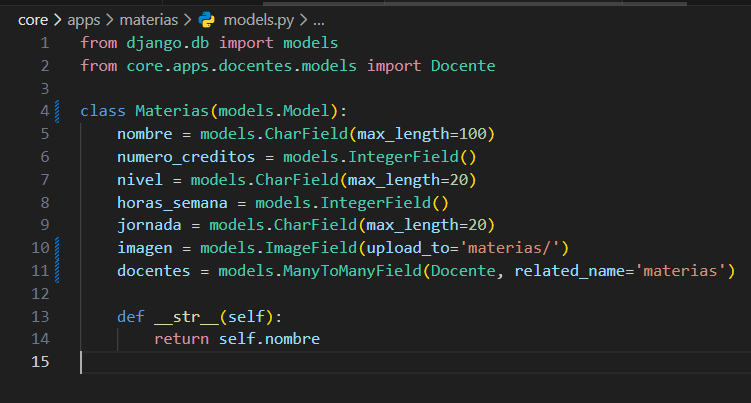
**Ilustración 9**: Modelo docente

### Modelo Materias

El modelo Curso representa las asignaturas ofertadas y contiene los siguientes atributos:

* nombre (CharField)
* numero\_creditos (IntegerField)
* nivel (CharField)
* horas\_semana (IntegerField)
* jornada (CharField)
* imagen (ImageField)

Este modelo representado en la aplicación se muestra a continuación :



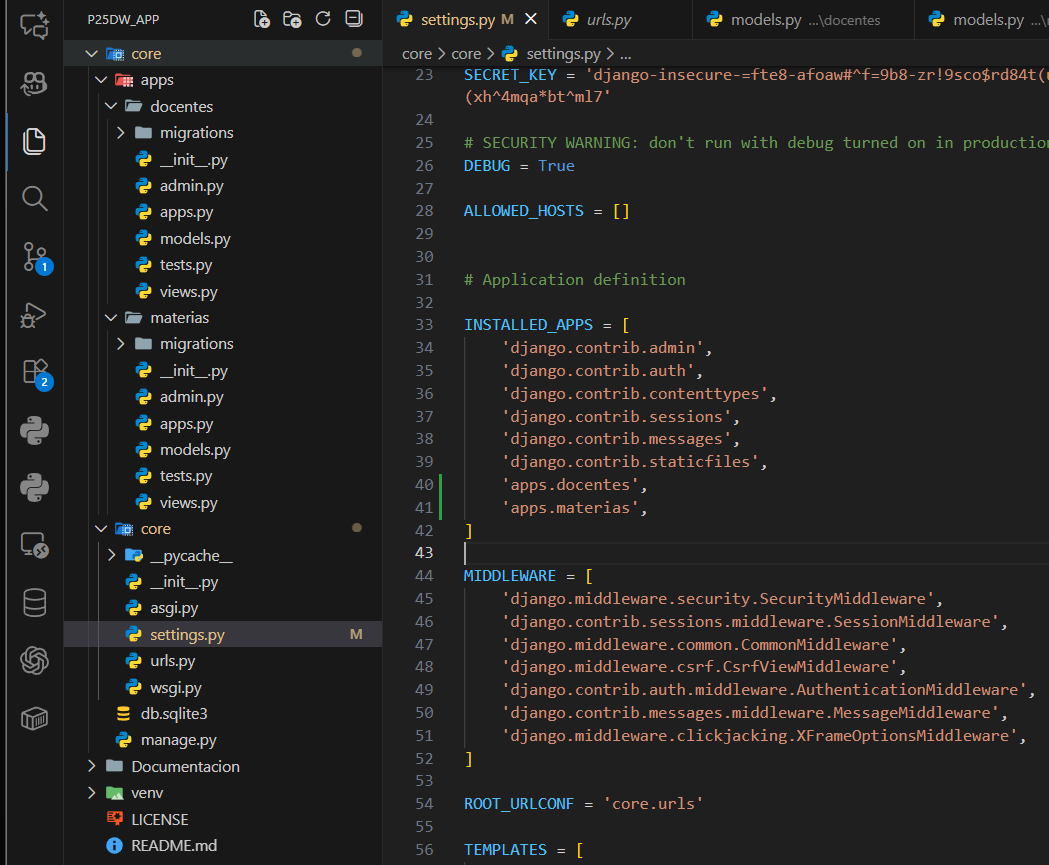
**Ilustración 10**: Modelo curso

## Migración y ORM

### Agregar aplicaciones en settings

Antes de realizar el proceso de migración utilizando el sistema de mapeo objeto-relacional (ORM) de Django, es fundamental registrar las aplicaciones creadas en el archivo settings.py. Este paso permite que el framework reconozca cada módulo y genere las tablas correspondientes en la base de datos.

Para ello, se accede al archivo settings.py ubicado en el directorio core/ y, en la sección INSTALLED\_APPS, se agregan las aplicaciones del proyecto, como se muestra a continuación:



**Ilustración 11**: Registro de aplicaciones en settings.py

### Generar las migraciones para cada modelo

Una vez definidos los modelos en las aplicaciones registradas, el siguiente paso consiste en generar los archivos de migración que contienen las instrucciones para crear las tablas en la base de datos. Este proceso se realiza mediante el comando:

python manage.py makemigrations



**Ilustración 12**: Creación de las migraciones

### Aplicar las migraciones

Después de generar las migraciones, es necesario aplicarlas para que los cambios definidos en los modelos se materialicen en la base de datos, esto permite crear o modificar las tablas según lo especificado en los archivos de migración.

Para ejecutar este proceso, se utiliza el siguiente comando:

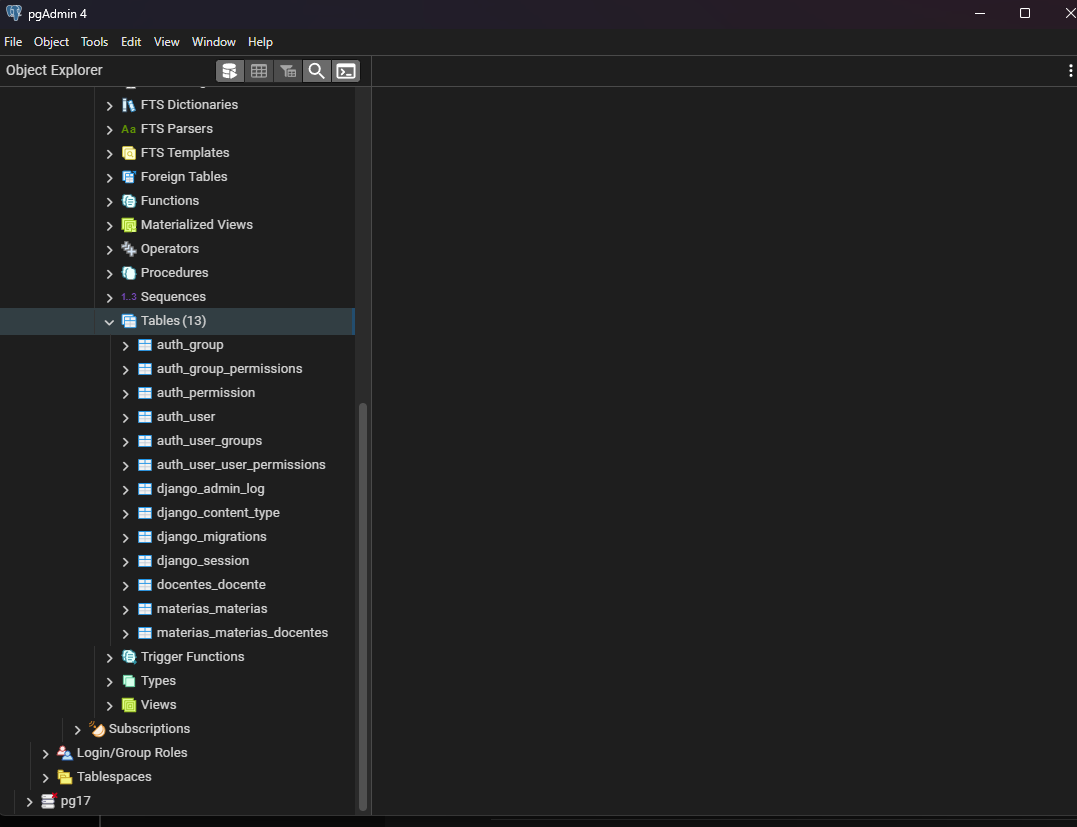
python manage.py migrate

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 13**: Importación de las migraciones

Finalmente, una vez aplicadas las migraciones, es importante verificar que las tablas hayan sido creadas correctamente en la base de datos.



**Ilustración 14**: Tablas en la base de datos Postgrest

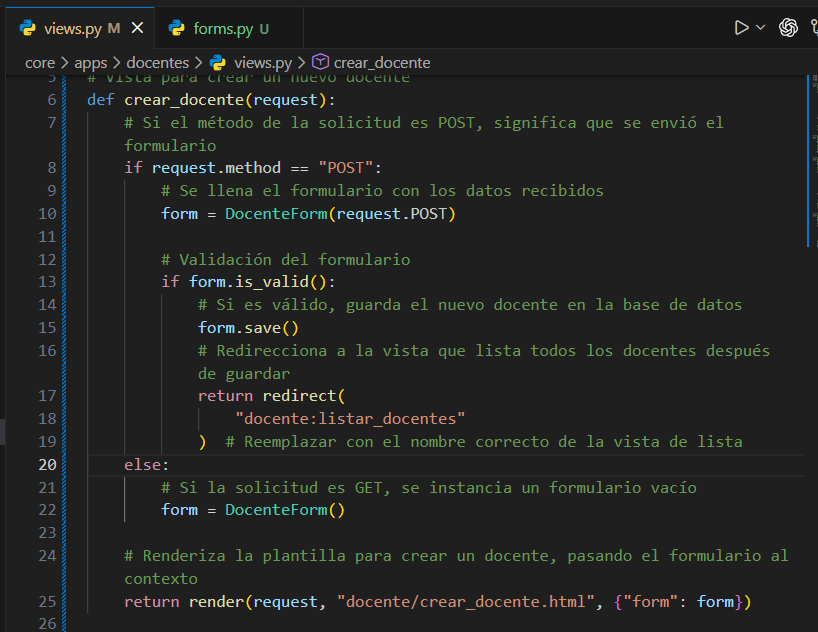
## Lógica de negocio

### CRUD docente

Las vistas en Django gestionan la lógica de interacción entre el usuario y los modelos de datos. Para el modelo Docente, se han implementado vistas específicas que permiten realizar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). A continuación, se describe cada una:

#### Vista Crear

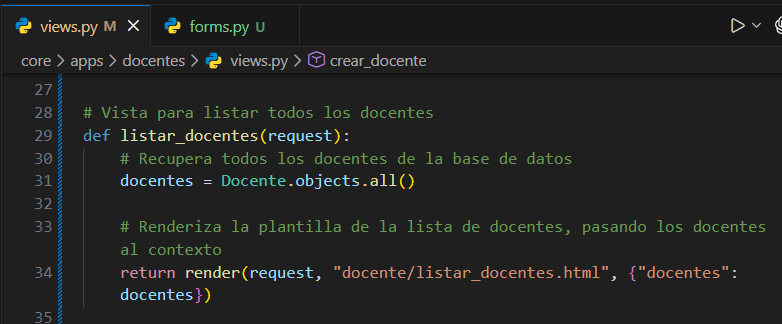
Esta vista permite registrar un nuevo docente en la base de datos, cuando el método de la solicitud es POST, se procesa el formulario con los datos enviados, se valida y, si es correcto, se guarda el registro en la base de datos, en caso de una solicitud GET, se muestra un formulario vacío para la creación.



**Ilustración 15**: función crear docente

#### Vista Leer

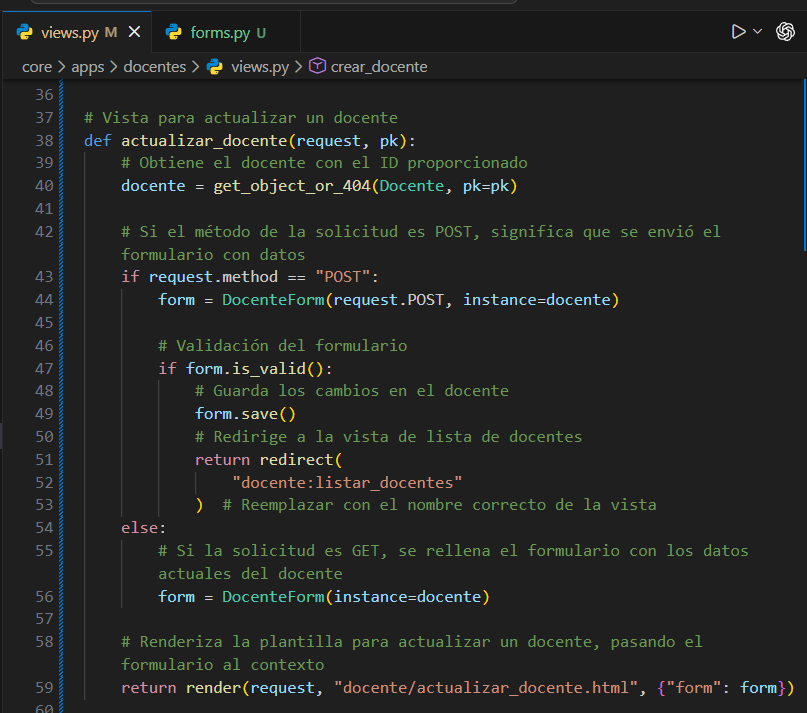
Esta vista recupera todos los registros del modelo Docente y los muestra en una lista o tabla para facilitar la visualización.



**Ilustración 16**: Función lista de docentes en la interfaz.

#### Vista Actualizar

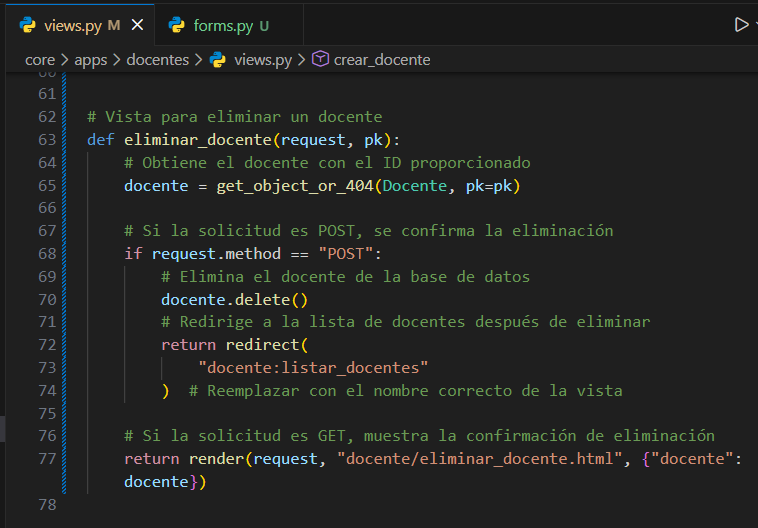
Permite editar los datos de un docente existente, se obtiene el registro mediante su ID, se carga en un formulario y, tras la validación, se guardan los cambios.



**Ilustración 19**: Captura del formulario con datos precargados.

#### Vista Eliminar

Gestiona la eliminación de un docente específico, antes de eliminar, se muestra una página de confirmación para evitar errores.



**Ilustración 20**: Función eliminar docente

#### Formulario para el Modelo

En este archivo se define el formulario que será utilizado para crear, actualizar y validar los datos del modelo Docente desde la interfaz web. El formulario actúa como intermediario entre el usuario y el modelo, permitiendo que la información ingresada en la vista se procese de manera segura y estructurada.

Para su implementación se utiliza la clase ModelForm proporcionada por Django, la cual permite generar automáticamente un formulario a partir de un modelo de base de datos. Este enfoque simplifica el desarrollo, ya que no es necesario declarar manualmente cada campo, reduciendo errores y asegurando la coherencia entre el formulario y el modelo definido.

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 21**: Formulario de docente

### CRUD Materias

En Django, las vistas son responsables de manejar la lógica entre el usuario y los modelos. Para el modelo Curso, se implementan vistas que permiten realizar las operaciones CRUD: Crear, Leer, Actualizar y Eliminar. A continuación, se describe cada funcionalidad con mayor detalle y ejemplos de código.

#### Vista Crear

Esta funcionalidad permite registrar una nueva materia en la base de datos. Cuando la solicitud es POST, se procesa el formulario con los datos enviados, se valida y, si todo es correcto, se guarda el registro. Si la solicitud es GET, se muestra un formulario vacío para que el usuario ingrese la información correspondiente.



**Ilustración 22**: Función crear materia

#### Vista Leer

La vista de lectura obtiene todas las materias almacenadas y las presenta en una lista organizada. Esta opción facilita la consulta de los registros y permite acceder a las acciones de edición o eliminación de manera sencilla.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Ilustración 23**: Función listar materia

#### Vista Actualizar

Esta vista permite modificar los datos de una materia existente. Se localiza el registro mediante su identificador único (pk), se carga en un formulario editable y, tras la validación, se guardan los cambios en la base de datos para mantener la información actualizada.

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 24**: Función actualizar materia

#### Vista Eliminar

Se encarga de eliminar una materia específica. Antes de realizar la acción, se muestra una página de confirmación para evitar errores y garantizar la integridad de los datos.

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 25**: Función eliminar materia

#### Formulario para el Modelo

. El formulario asociado al modelo Materias se implementa mediante la clase ModelForm de Django, lo que permite generar automáticamente los campos a partir del modelo. Este método agiliza el desarrollo y asegura la coherencia entre el formulario y la estructura de la base de datos.

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 26**: Formulario de la materia

## Diseño templates

El diseño de plantillas representa un elemento clave en la construcción de la interfaz gráfica del sistema, ya que define cómo se presentan las funcionalidades al usuario de manera clara y organizada. A través del motor de plantillas de Django, se implementarán vistas HTML dinámicas que no solo permitan mostrar información, sino también interactuar con las operaciones principales del sistema, como la creación, listado, actualización y eliminación de registros.

Por otro lado, la organización de las plantillas se basará en una estructura común que incluirá componentes compartidos como la cabecera, el pie de página y la barra de navegación, garantizando consistencia visual en todo el sistema. Además, se aplicará un diseño responsivo que permita la correcta adaptación de la interfaz en diferentes dispositivos, priorizando la legibilidad, la accesibilidad y la facilidad de uso.

### Barra de navegación

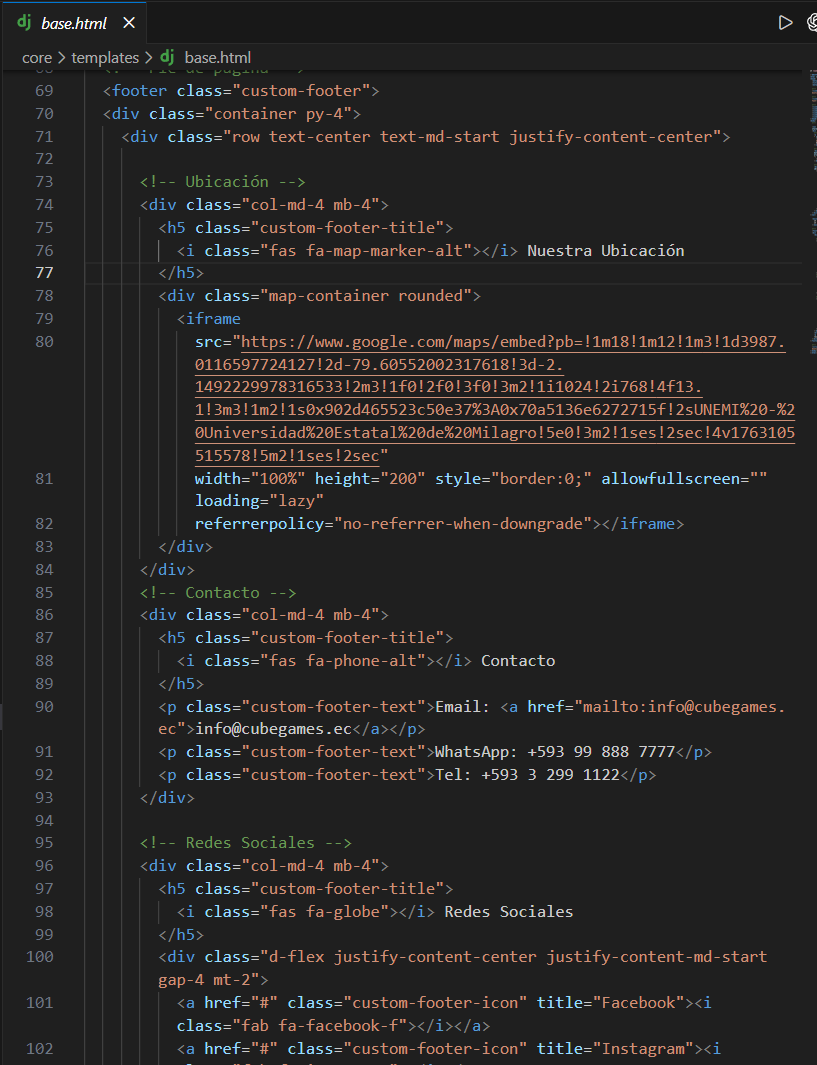
La barra de navegación representa un componente esencial del sitio web, ya que permite acceder de manera organizada a las diferentes secciones; en este caso, se ha implementado para el portal de oferta de cursos de la UNEMI utilizando HTML y Bootstrap con el fin de garantizar una estructura responsiva y funcional, incorporando el logotipo y título del sitio junto con un ícono representativo, un botón tipo hamburguesa para dispositivos móviles y un menú que incluye las opciones Home, Docentes y Cursos, todo ello con un diseño que combina colores institucionales y estilos personalizados para mejorar la experiencia del usuario.



**Ilustración 27**: Código html de navbar

### Footer

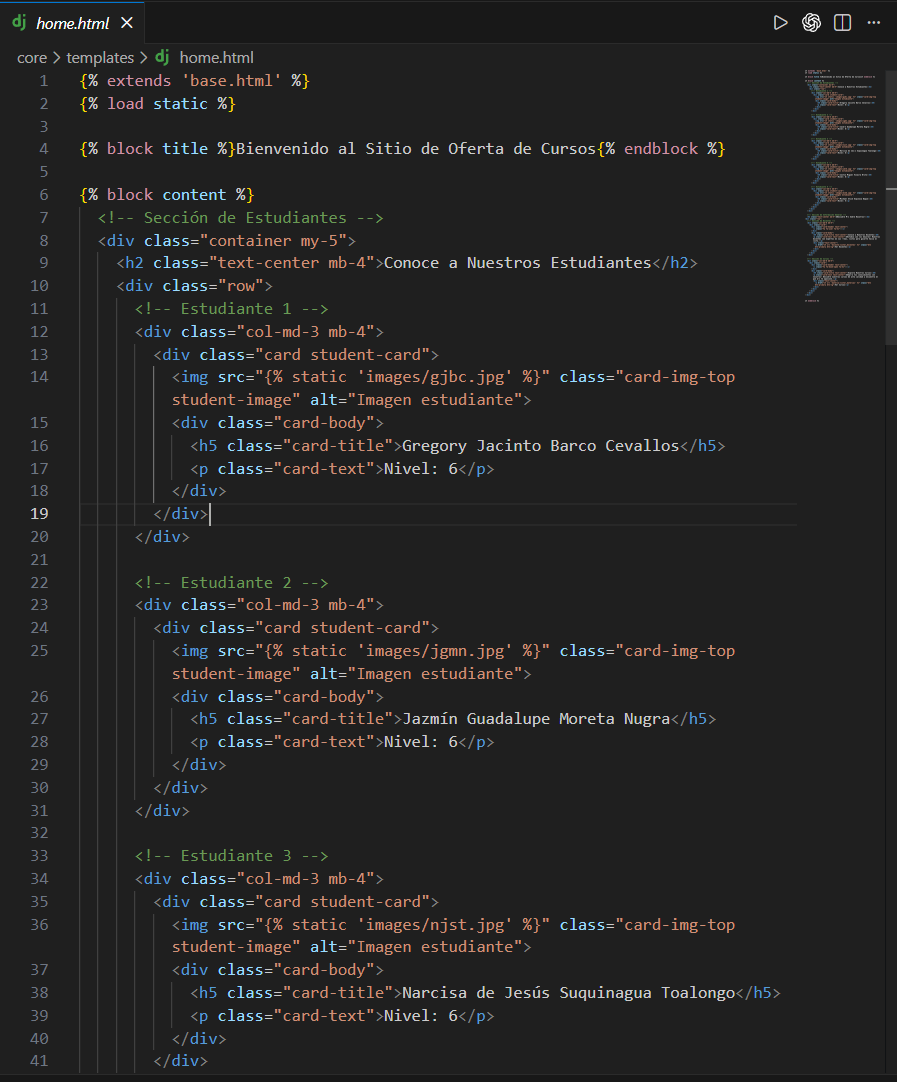
El pie de página se ha implementado como un componente estructural que proporciona información complementaria y enlaces relevantes, utilizando HTML y clases personalizadas de Bootstrap para garantizar una disposición responsiva y ordenada; este elemento incluye tres secciones principales: la primera muestra la ubicación institucional mediante un mapa incrustado de Google Maps dentro de un contenedor adaptable, la segunda presenta los datos de contacto con enlaces funcionales para correo electrónico y números telefónicos, y la tercera integra íconos interactivos que enlazan a redes sociales, todo ello acompañado de un divisor horizontal y un texto centrado con derechos reservados, asegurando así una experiencia coherente y accesible para el usuario.



**Ilustración 28**: Código html de footer

1. *Página principal*

La página principal se estructura como un módulo central que organiza contenido informativo y elementos visuales mediante un diseño responsivo basado en Bootstrap, incorporando dos bloques funcionales claramente diferenciados; el primero corresponde a la sección de estudiantes, donde se emplea un sistema de tarjetas con imágenes, nombres y niveles académicos, distribuidas en una cuadrícula adaptable para optimizar la visualización en distintos tamaños de pantalla, mientras que el segundo bloque agrupa información general sobre docentes y cursos mediante componentes tipo card que integran encabezados, íconos representativos, descripciones breves y botones con enlaces dinámicos



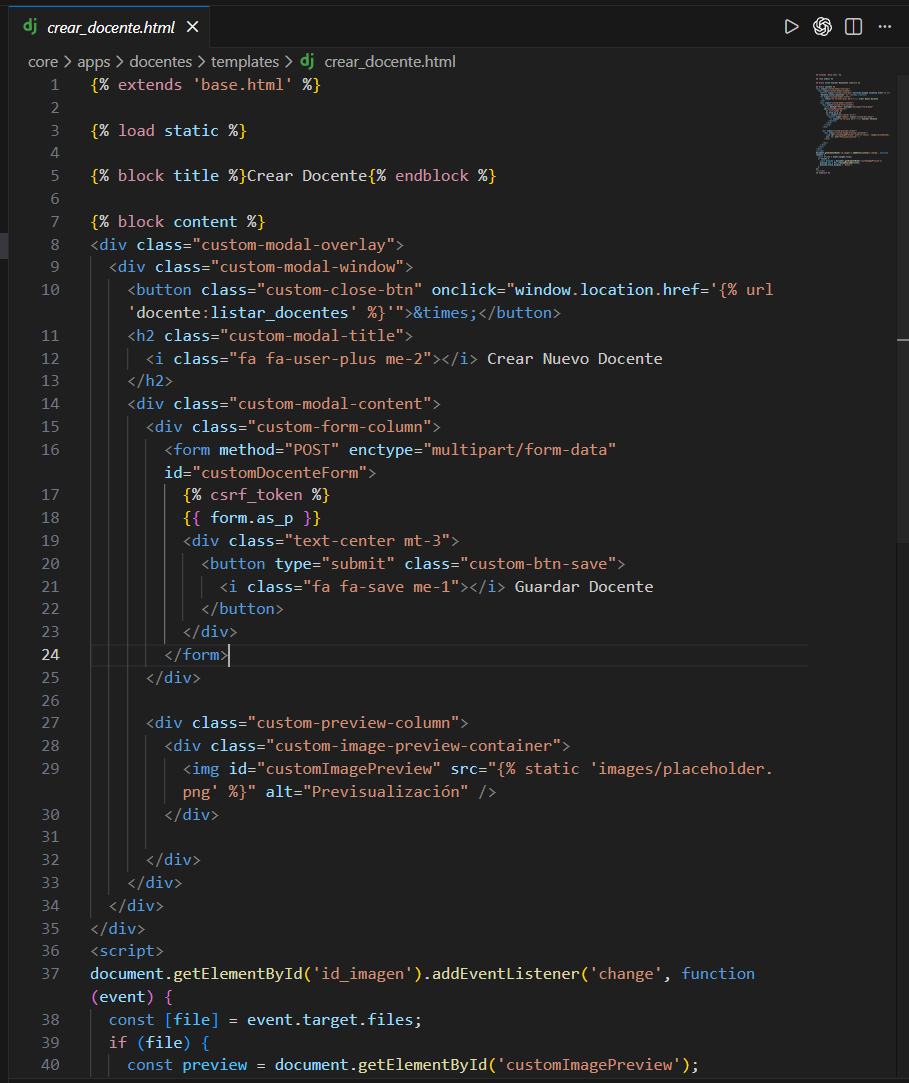
**Ilustración 29**: Código html de página principal

### Docente

#### Agregar docente

La funcionalidad para agregar docentes se implementa mediante una plantilla que extiende la estructura base definida en base.html utilizando la etiqueta {% extends 'base.html' %}, lo que permite mantener la coherencia visual en todo el sitio; adicionalmente, se carga el módulo de archivos estáticos con {% load static %} para gestionar recursos como imágenes y estilos. Dentro del bloque {% block content %} se define un contenedor principal con clases personalizadas (custom-modal-overlay y custom-modal-window) que simulan una ventana modal, incorporando un botón de cierre con evento onclick que redirige a la vista de listado de docentes mediante la etiqueta {% url 'docente:listar\_docentes' %}.

El formulario se declara con el atributo method="POST" y enctype="multipart/form-data" para permitir la carga de archivos, incluyendo la protección CSRF con {% csrf\_token %} y la representación dinámica de los campos mediante {{ form.as\_p }}. Se añade un botón de envío con ícono Font Awesome (fa-save) y clases personalizadas para estilos. Paralelamente, se implementa una columna de previsualización que utiliza la etiqueta <img> con un recurso estático como imagen por defecto y un script en JavaScript que escucha el evento change del input id\_imagen para actualizar la previsualización utilizando URL.createObjectURL(file), garantizando así una experiencia interactiva y funcional para el usuario.

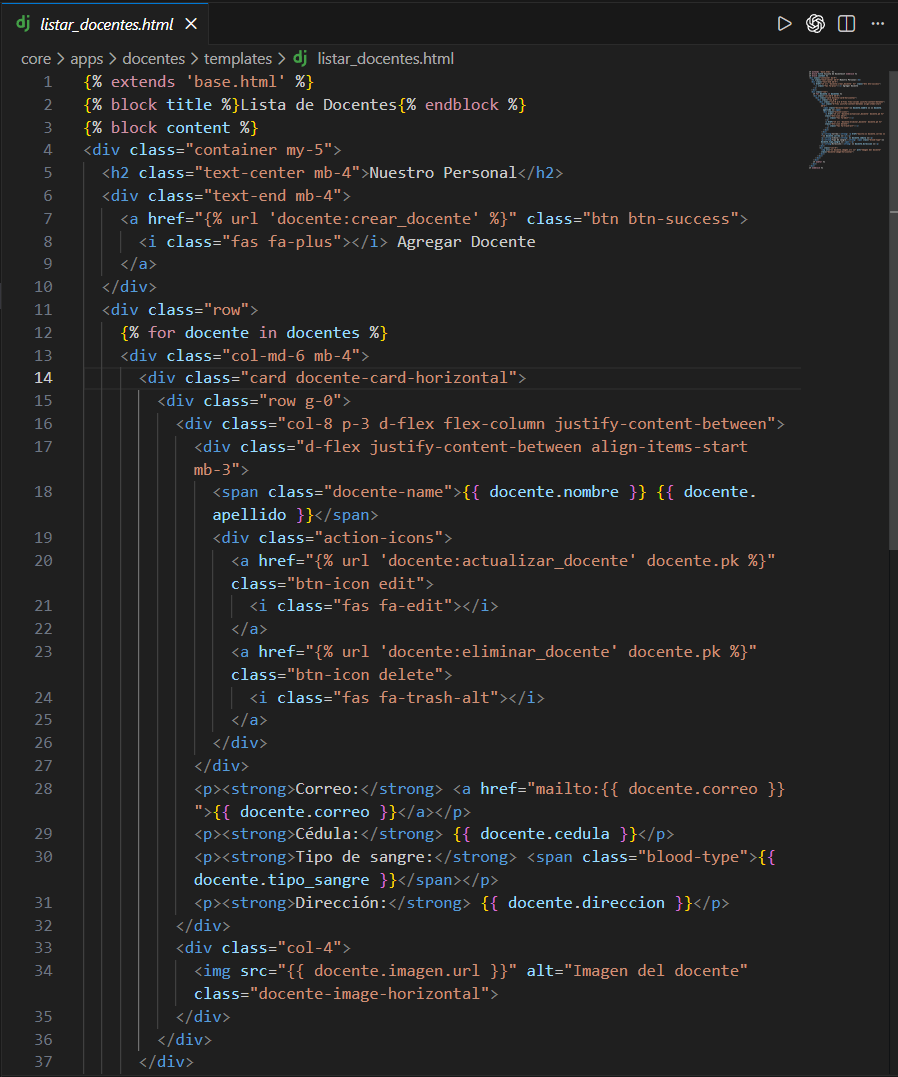


**Ilustración 30**: Código html de crear docente

#### Listar docentes

E listado de docentes se desarrollo sobre una plantilla que hereda la estructura global mediante la directiva {% extends 'base.html' %}, lo que permite mantener consistencia en el diseño y reutilización de componentes comunes; se define el título dinámico dentro del bloque {% block title %} y el contenido principal en {% block content %}.

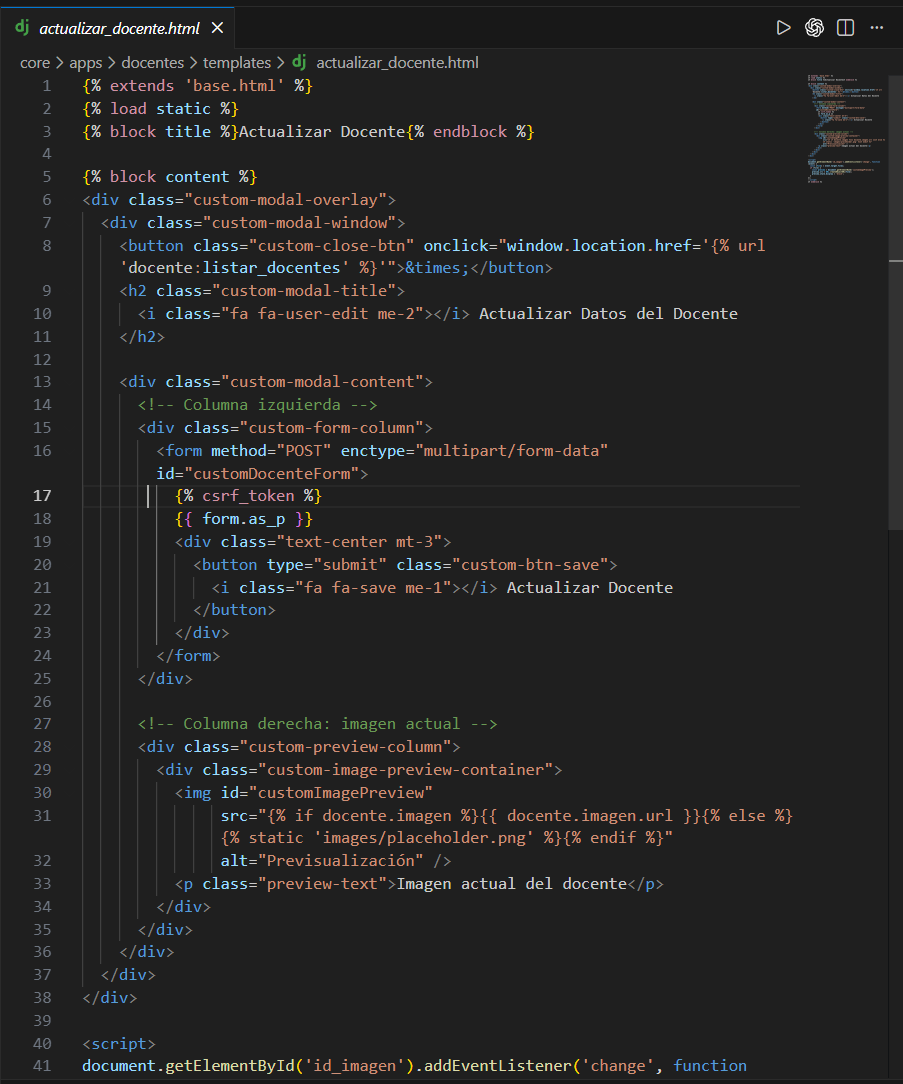
El layout utiliza un contenedor <div class="container my-5"> para aplicar espaciado vertical y organizar los elementos, seguido de un encabezado <h2> centrado y un botón de acción con clases Bootstrap (btn btn-success) que invoca la vista de creación mediante la etiqueta {% url 'docente:crear\_docente' %}, además el listado se genera con un bucle {% for docente in docentes %} que recorre la colección enviada desde la vista, renderizando tarjetas horizontales (card docente-card-horizontal) dentro de una estructura <div class="row g-0"> que elimina el espacio interno entre columnas.



**Ilustración 31**: Código html de listar docente

#### Editar docente

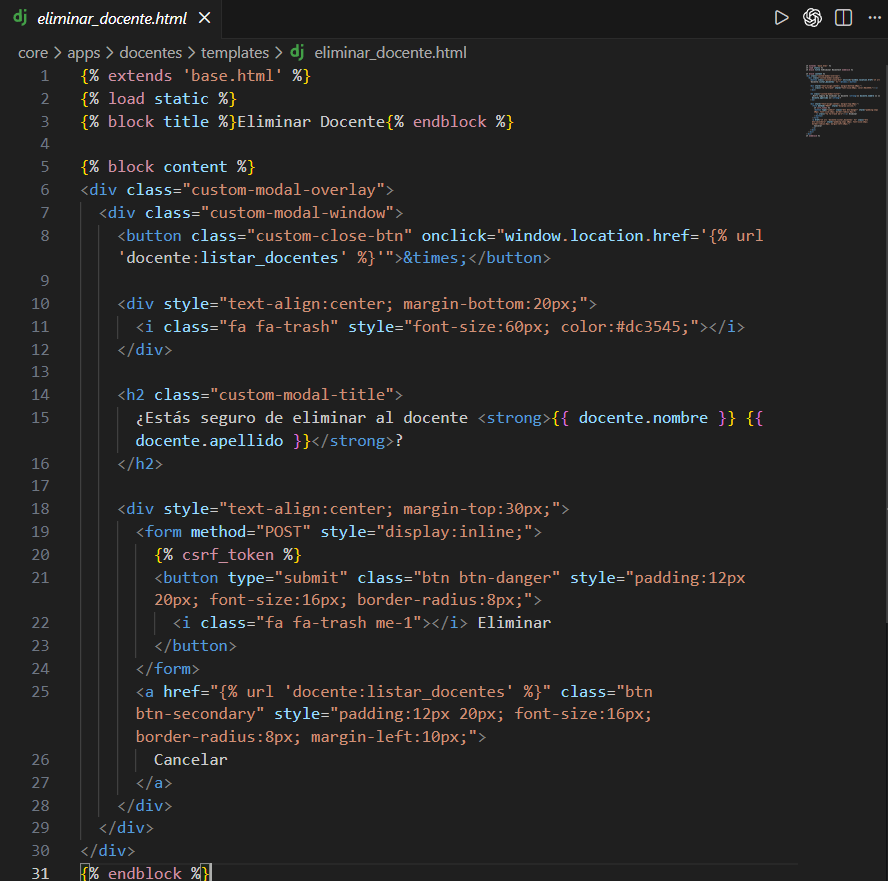
La interfaz para editar docentes se organiza en una ventana modal personalizada que contiene dos columnas: la primera incluye el formulario con método POST, soporte para carga de archivos (enctype="multipart/form-data") y protección CSRF, renderizando los campos mediante {{ form.as\_p }}; la segunda muestra la imagen actual del docente utilizando una condición en la plantilla para verificar si existe un archivo asociado, en cuyo caso se carga con {{ docente.imagen.url }}, o se muestra una imagen por defecto desde los recursos estáticos. Se incorpora un botón de actualización con ícono Font Awesome y un script en JavaScript que gestiona la previsualización dinámica de la imagen seleccionada mediante URL.createObjectURL(file), garantizando una experiencia interactiva y coherente con el diseño responsivo.



**Ilustración 32**: Código html de actualizar docente

#### Eliminar docente

La acción de eliminación se gestiona mediante una ventana modal que solicita confirmación antes de ejecutar el proceso, mostrando el nombre completo del docente y un ícono representativo para alertar al usuario; el diseño incluye un formulario con método POST y protección CSRF para garantizar seguridad, acompañado de un botón principal con estilo btn-danger que confirma la eliminación y un enlace alternativo para cancelar la operación, evitando acciones accidentales. Esta estructura asegura una interacción clara y segura, reforzada por la lógica condicional en la plantilla que mantiene la coherencia visual y funcional del sistema.

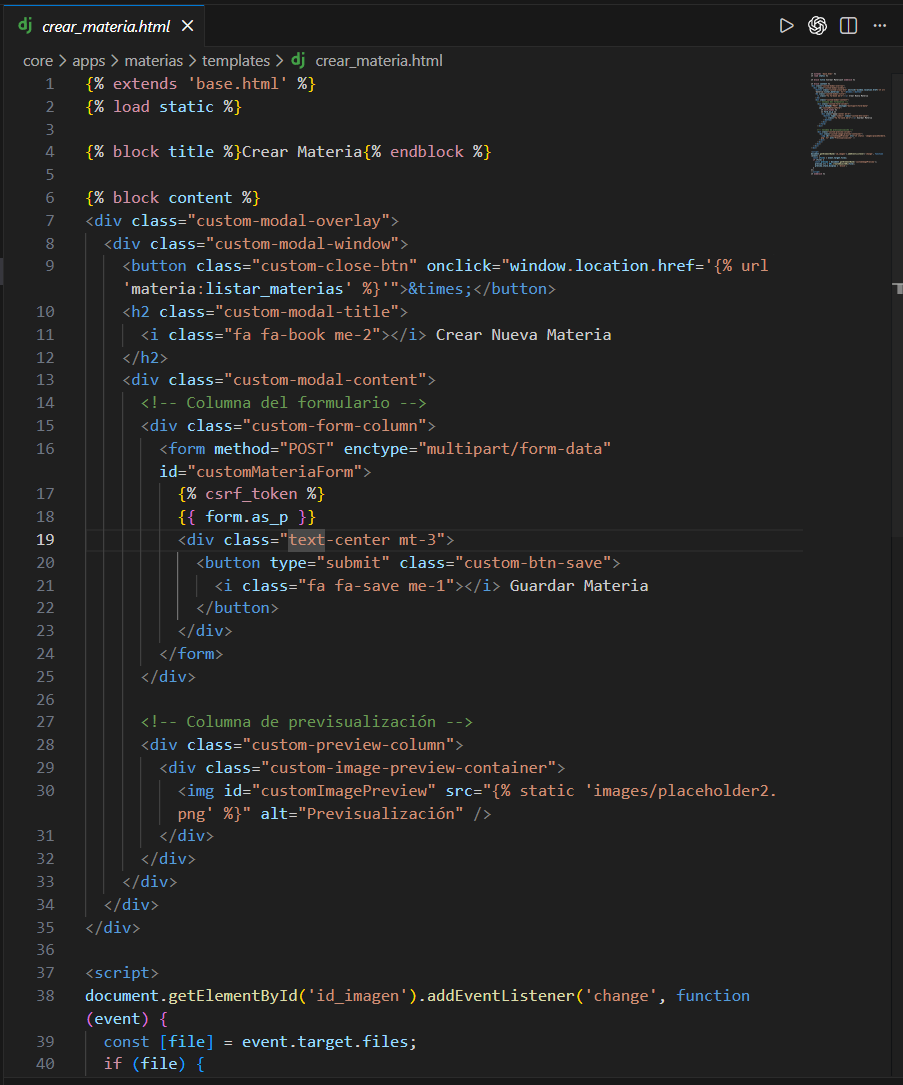


**Ilustración 33**: Código html de eliminar docente

### Materias

#### Agregar materias

La funcionalidad para crear materias se presenta en una ventana modal compuesta por dos columnas: la primera contiene el formulario con método POST, soporte para carga de archivos y protección CSRF, renderizado mediante {{ form.as\_p }} para simplificar la estructura; la segunda incluye un área de previsualización con una imagen por defecto cargada desde los recursos estáticos. Se incorpora un botón de acción con ícono Font Awesome y un script en JavaScript que actualiza dinámicamente la imagen seleccionada utilizando URL.createObjectURL(file), garantizando una experiencia interactiva y coherente con el diseño responsivo.



**Ilustración 34**: Código html de agregar curso

#### Lista de materias

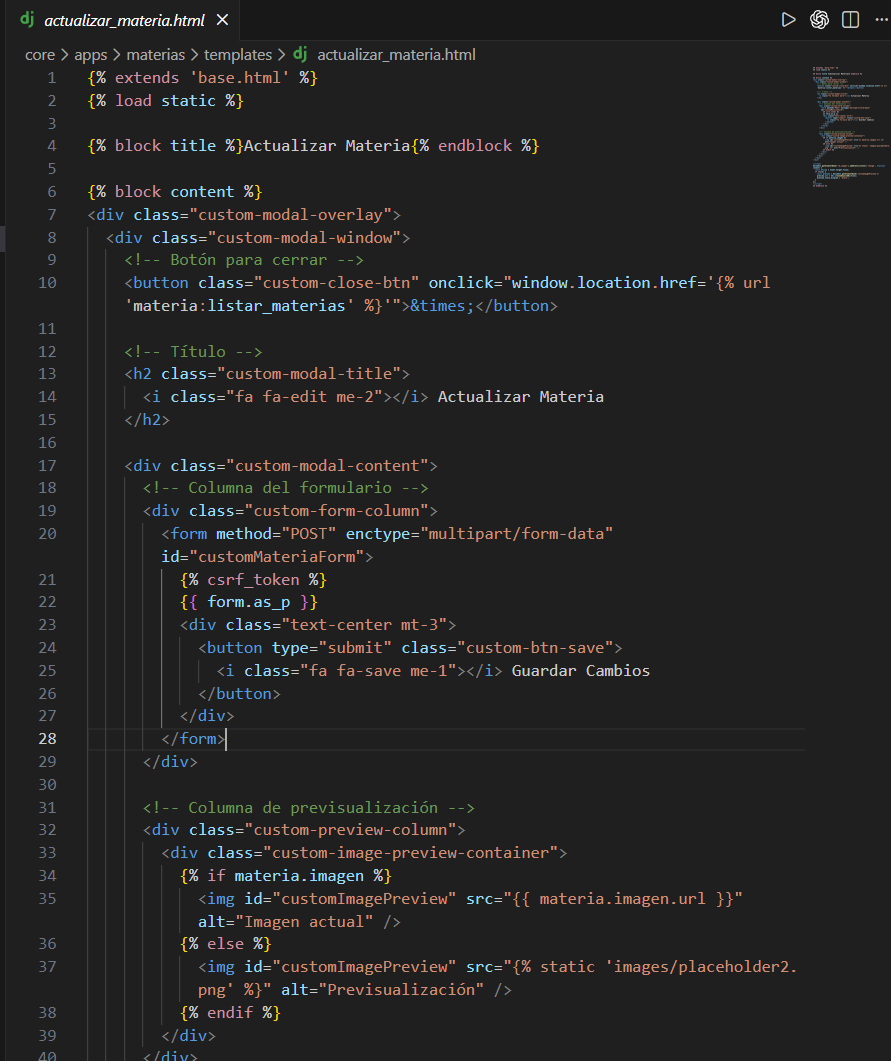
La vista de listado de materias se organiza en un contenedor principal con clases de Bootstrap para mantener la estructura responsiva, incorporando un encabezado centrado y un botón de acción que enlaza a la creación de nuevas materias mediante la etiqueta {% url 'materia:crear\_materia' %}. El contenido dinámico se genera con un bucle {% for materia in materias %}, que renderiza tarjetas horizontales con dos secciones: la primera muestra el nombre de la materia, docentes asociados y datos relevantes como nivel, créditos, horas y jornada, utilizando badges para resaltar la información; la segunda presenta la imagen del curso, aplicando una condición para mostrar la imagen cargada o una por defecto desde los recursos estáticos. Además, se incluyen íconos interactivos para editar y eliminar, con enlaces construidos mediante las URLs dinámicas de Django, garantizando una interfaz clara y funcional.



**Ilustración 35**: Código html de listar cursos

#### Editar materia

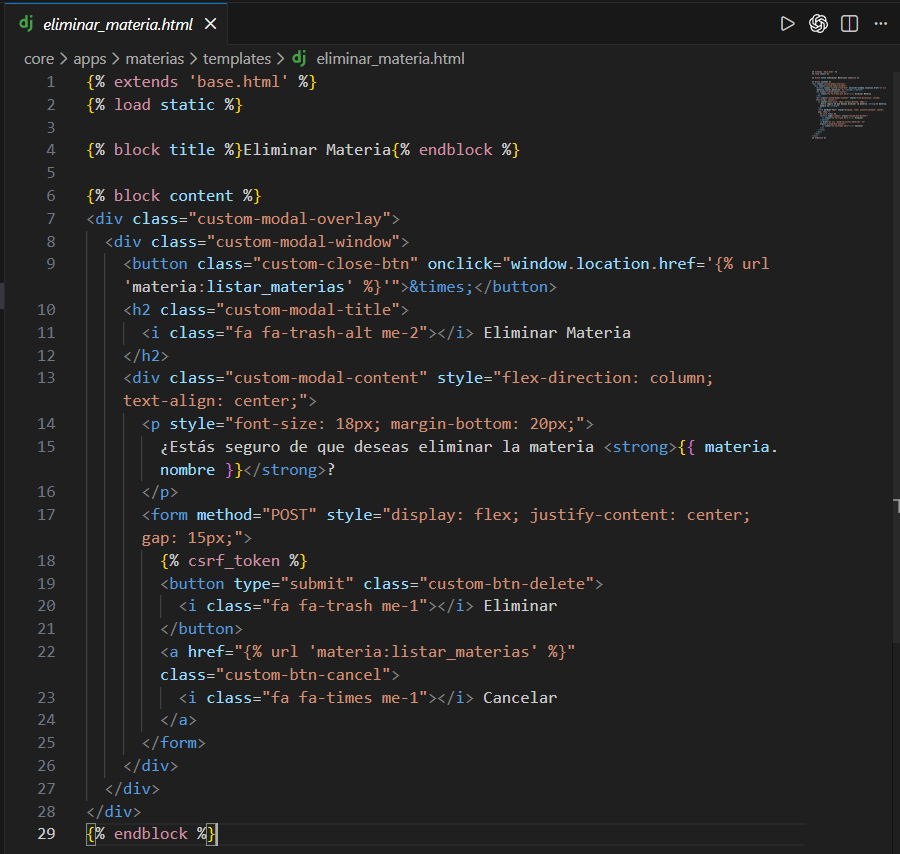
La interfaz para actualizar materias se organiza en una ventana modal con dos columnas: la primera contiene el formulario con método POST, soporte para carga de archivos y protección CSRF, renderizado mediante {{ form.as\_p }} para simplificar la estructura; la segunda muestra la imagen actual de la materia utilizando una condición que verifica si existe un archivo asociado, en cuyo caso se carga desde {{ materia.imagen.url }}, o se presenta una imagen por defecto desde los recursos estáticos. Se incluye un botón de acción con ícono Font Awesome y un script en JavaScript que permite la previsualización dinámica de la nueva imagen seleccionada mediante URL.createObjectURL(file), garantizando una experiencia interactiva y coherente con el diseño responsivo.



**Ilustración 36**: Código html de editar cursos

#### Eliminar materia

La eliminación de materias se gestiona mediante una ventana modal que solicita confirmación antes de ejecutar la acción, mostrando el nombre de la materia y un ícono de advertencia para reforzar la intención del proceso. El diseño incluye un formulario con método POST y protección CSRF, acompañado de dos botones: uno principal con estilo custom-btn-delete para confirmar la eliminación y otro secundario para cancelar, evitando errores por acciones involuntarias.



**Ilustración 37**: Código html de eliminar cursos

# Resultados

### Navbar



**Ilustración 38**: Interfaz resultado navbar

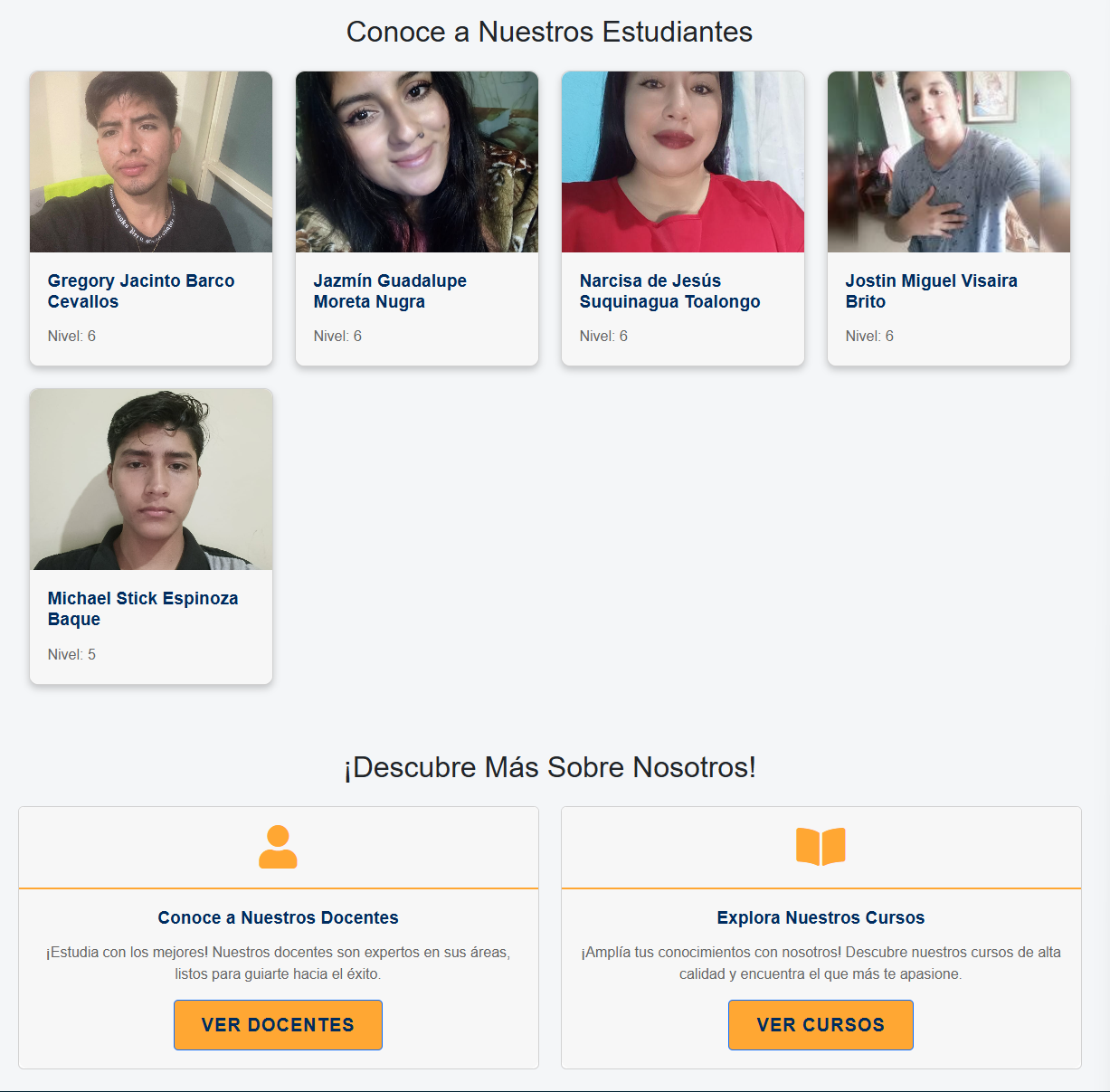
### Footer

Texto

El contenido generado por IA puede ser incorrecto.

**Ilustración 39**: Interfaz resultado footer

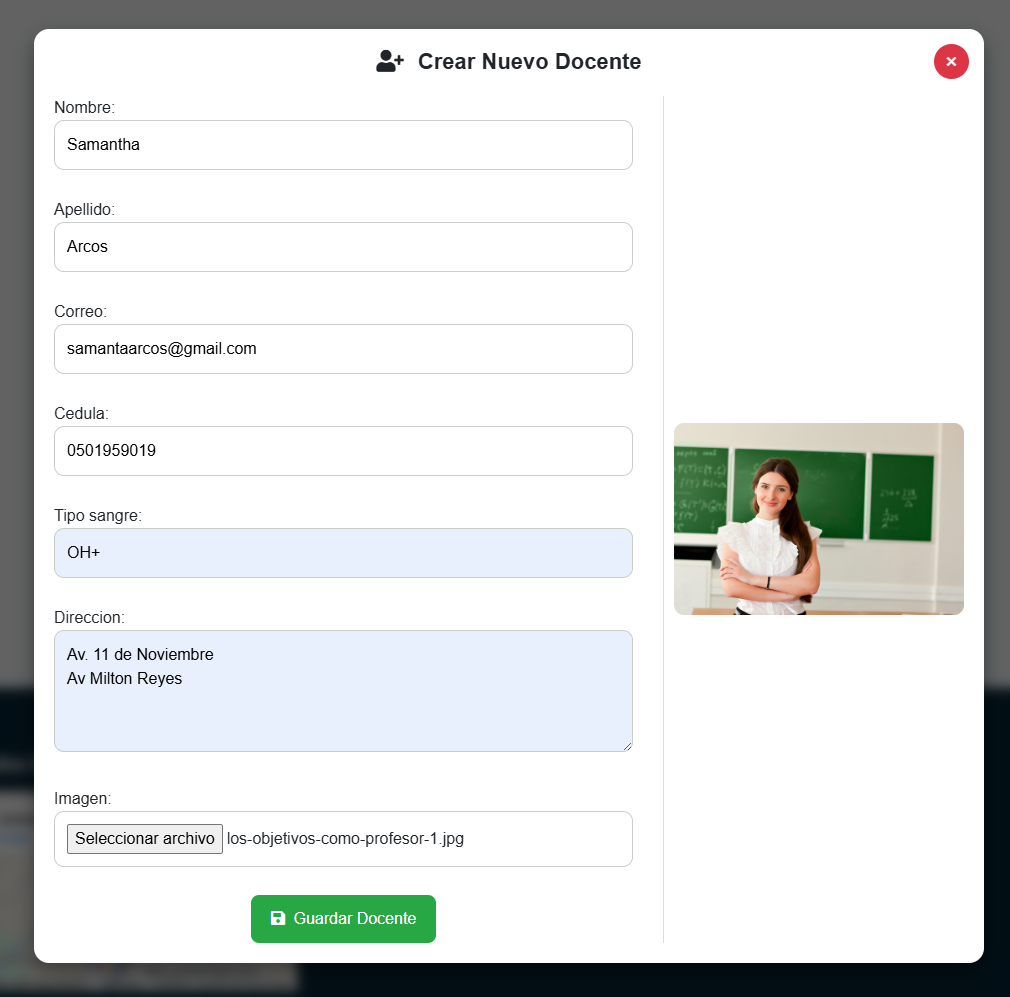
### Página principal



**Ilustración 40**: Interfaz resultado página principal

### Docentes

* agregar docente



**Ilustración 41**: Interfaz resultado agregar docente

* Vista principal

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Ilustración 42**: Interfaz resultado lista docentes

* Editar docente

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Ilustración 42:** Interfaz resultado editar docentes

* Eliminar docente

**Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.**

**Ilustración 43:** Interfaz resultado editar docentes

### Materias

* Agregar curso

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Ilustración 44:** Interfaz resultado agregar curso

* Lista de cursos

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Ilustración 45:** Interfaz resultado lista curso

* Editar curso

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Ilustración 46:** Interfaz resultado editar curso

* Eliminar producto

**Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.**

**Ilustración 47:** Interfaz resultado editar curso

# Conclusiones

El desarrollo del proyecto permitió configurar de manera correcta y eficiente un entorno completo basado en Django y PostgreSQL, cumpliendo con todos los objetivos planteados, la creación del entorno virtual, la instalación de dependencias, la configuración del framework y la conexión con la base de datos mediante psycopg2 aseguraron una estructura sólida y profesional para el sistema. Asimismo, la definición clara de los modelos y la implementación de la arquitectura MTV facilitaron una organización adecuada del código, garantizando la escalabilidad, la mantenibilidad y la coherencia interna de la aplicación.

La implementación de las operaciones CRUD para las aplicaciones Docentes y Materias demostró la funcionalidad del ORM de Django y la capacidad del framework para gestionar datos de manera eficiente, cada una de las vistas construidas permitió realizar procesos de creación, lectura, actualización y eliminación de forma segura y estructurada, paralelamente, los formularios basados en ModelForm aseguraron la validación y el manejo adecuado de los datos, mientras que las rutas y plantillas integraron de manera correcta todos los componentes del sistema en una experiencia fluida para el usuario.

Finalmente, el diseño de las interfaces mediante HTML, CSS, Bootstrap y el sistema de plantillas de Django resultó en una plataforma visualmente organizada, responsiva y fácil de usar. La creación del navbar, footer, página principal y las vistas específicas para docentes y materias evidencia una comprensión sólida de las buenas prácticas en desarrollo web, en conjunto, el proyecto cumple satisfactoriamente con la propuesta inicial, integrando modelos de datos, lógica de negocio y diseño visual en una solución completa, funcional y alineada con los estándares profesionales del desarrollo de aplicaciones web modernas.

# Bibliografía

Fowler, M. (2018). Refactoring: Improving the design of existing code (2nd ed.). Addison-Wesley.

Kennedy, J., & Fogel, J. (2017). psycopg2: PostgreSQL database adapter for Python. Recuperado de <https://www.psycopg.org/docs/>

Krug, S. (2014). Don’t make me think, revisited: A common sense approach to web usability (3rd ed.). New Riders.

Momjian, B. (2021). PostgreSQL concepts and future challenges. Recuperado de <https://momjian.us/main/blogs/pgblog/>

Sweigart, A. (2020). Automate the boring stuff with Python (2nd ed.). No Starch Press.